第9章 软件架构的演化和维护

第9章 软件架构的演化和维护

- 软件架构的生命周期
  - 初始设计
  - 实际使用
  - 修改完善（软件架构的演化与维护过程）
  - 退化弃用
- 软件架构演化就是为了维持软件架构自身的有用性。

第9章 软件架构的演化和维护

- 9.1软件架构演化和软件架构定义的关系
- 9.2软件架构演化方式的分类
- 9.3软件架构演化原则
- 9.4软件架构维护
- 9.5本章小结

9.1软件架构演化和软件架构定义的关系

- 我们在理解软件架构演化的时候，需要考虑具体的软件架构定义。
  - 例如，如果软件架构定义是

$$SA = \{components, connectors, constraints\}$$

  - 也就是说，软件架构包括组件（components）、连接件（connectors）和约束（constraints）三大要素，这类软件架构演化主要关注的就是组件、连接件和约束的添加、修改与删除等。

9.1软件架构演化和软件架构定义的关系

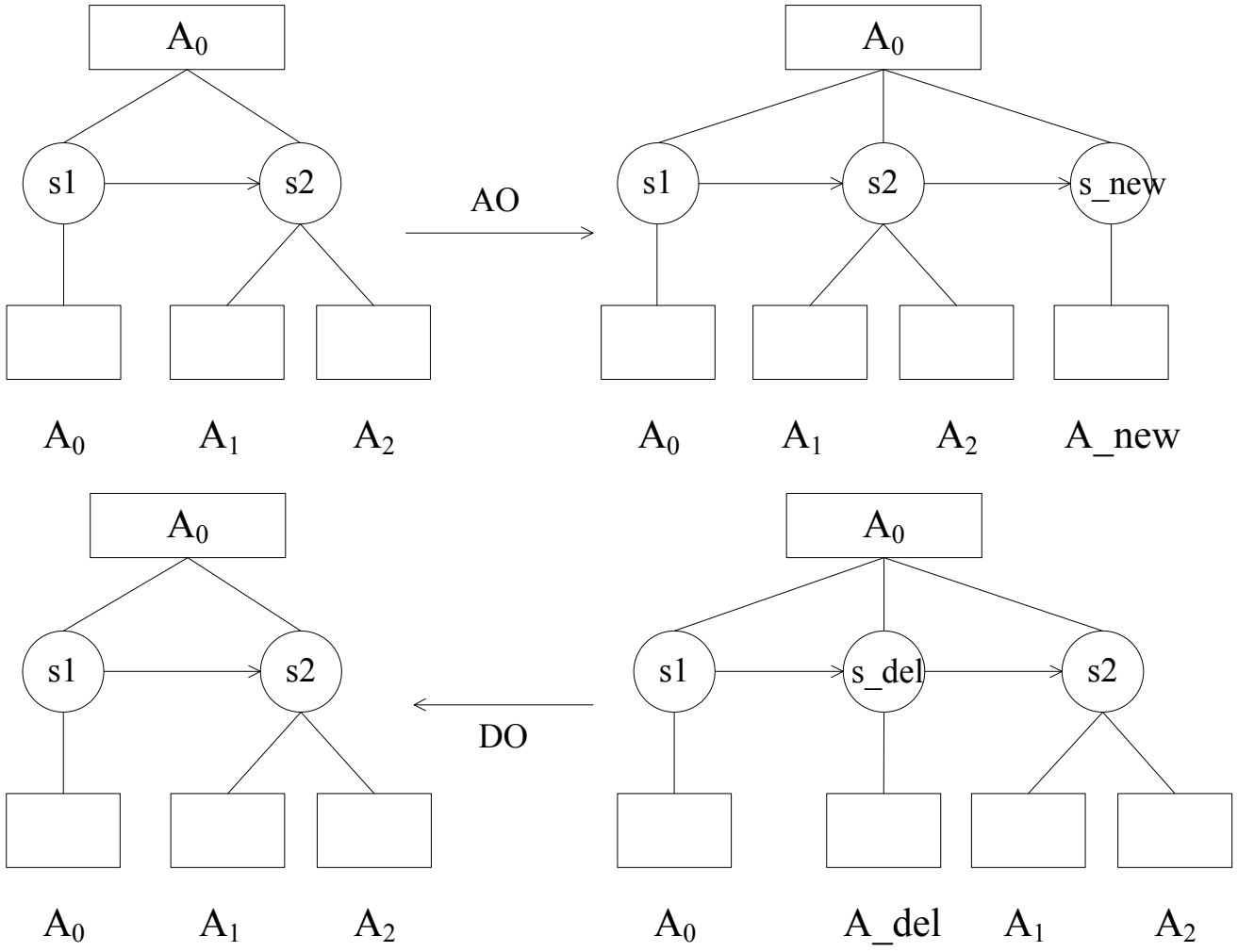
- 我们以面向对象软件架构（OOSA，object-oriented software architecture）为例，结合UML顺序图来进一步讨论各种演化操作，利用层次自动机给出各个演化操作的具体演化规则。

9.1软件架构演化和软件架构定义的关系

- （1）对象演化
  - 在顺序图中，组件的实体为对象。
  - 组件本身包含了众多的属性，如接口、类型、语义等等，这些属性的演化是对象自身的演化，对于描述对象之间的交互过程并无影响。
  - 因此，会对架构设计的动态行为产生影响的演化只包括Add Object和Delete Object两种

9.1软件架构演化和软件架构定义的关系

- 对象演化的自动机表示



9.1软件架构演化和软件架构定义的关系

- （2）消息演化
  - 消息是顺序图中的核心元素，包含了名称、源对象、目标对象、时序等等信息。
  - 这些信息与其他对象或消息相关联，产生的变化会直接影响到对象之间的交互，从而对架构的正确性或时态属性产生影响。
  - 消息自身的属性，如接口、类型等等，产生的变化不会影响到对象之间交互的过程，则不考虑其发生的演化类型。
  - 因此，将消息演化分为Add Message，Delete Message，Swap Message Order（交换消息时间顺序），Overturn Message（消息角色反转），Change Message Module（改变消息发送或接受对象）五种

9.1软件架构演化和软件架构定义的关系

- （3）复合片段演化
  - 复合片段是对象交互关系的控制流描述，表示可能发生在不同场合的交互，与消息同属于连接件范畴。
  - 复合片段本身的信息包括类型、成立条件和内部执行序列，其中内部执行序列的演化等价于消息序列演化。
  - 会产生分支的复合片段包括ref、loop、break、alt、opt、par（产生并行消息），主要考虑这些会产生分支的复合片段所产生的演化。
  - 将复合片段的演化分为Add Fragment，Delete Fragment，Type Change（改变复合片段的类型）。

9.1软件架构演化和软件架构定义的关系

- （4）约束演化
  - 顺序图中的约束信息以文字描述的方式存储于对象或消息中，例如通常可以用LTL来描述时态属性约束。
  - 约束演化对应着架构配置的演化，一般来源于系统属性的改变，而更多情况下约束会伴随着消息的改变而发生改变。
  - 约束演化即直接对约束信息进行添加、删除。

9.2软件架构演化方式的分类

- 针对软件架构的演化过程是否处于系统运行时期，可以将软件架构演化分为
  - 软件架构静态演化
    - 发生在软件架构的设计、实现和维护过程中，软件系统还未运行或者处在运行停止状态
  - 软件架构动态演化
    - 发生在软件系统运行过程中

9.2 软件架构演化方式的分类

- 1. 软件架构静态演化

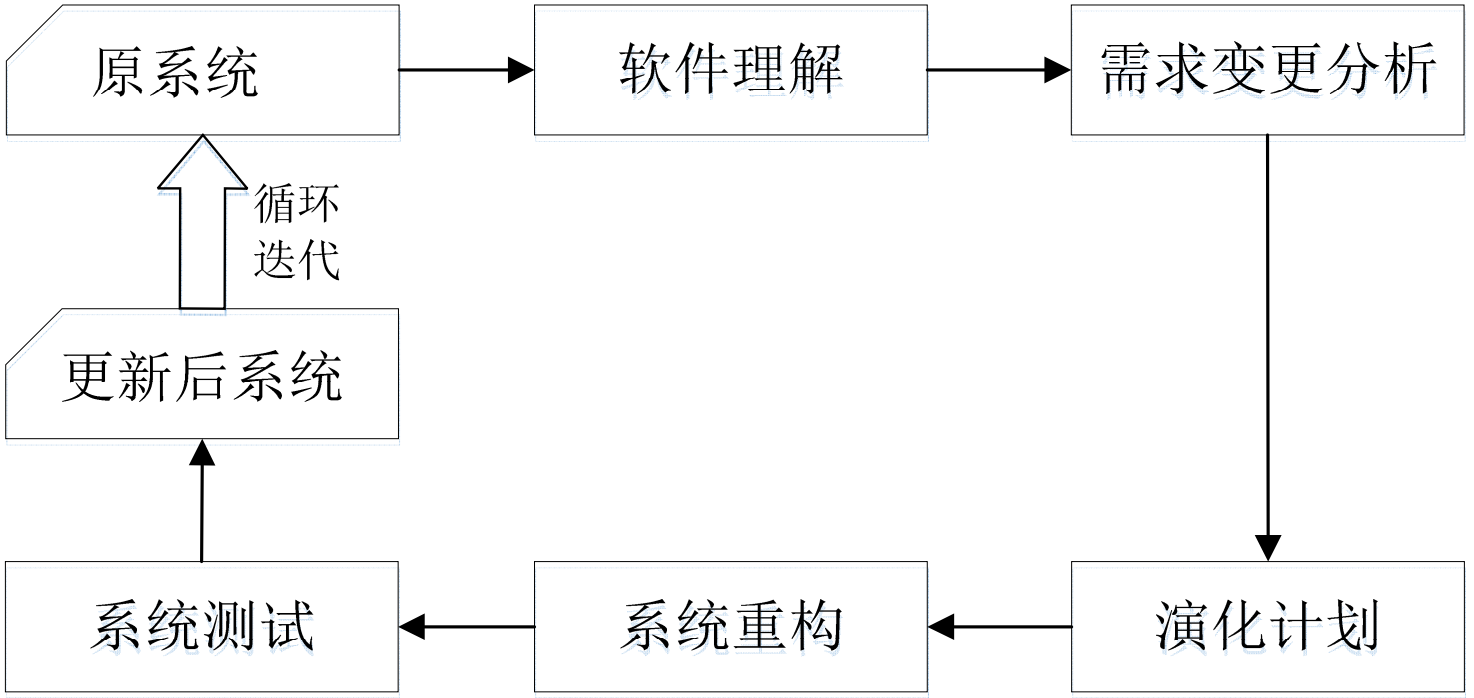
- 静态演化需求

（1）设计时演化需求：在架构开发和实现过程中对原有架构进行调整，保证软件实现与架构的一致性以及软件开发过程的顺利进行；

（2）运行前演化需求：软件发布之后由于运行环境的变化，需要对软件进行修改升级，在此期间软件的架构同样要进行演化。

9.2软件架构演化方式的分类

- 1. 软件架构静态演化
- **静态演化一般过程**



9.2软件架构演化方式的分类

- 1.软件架构静态演化

- 静态演化一般过程

（1）软件理解：查阅软件文档，分析软件架构，识别系统组成元素及其之间的相互关系，提取系统的抽象表示形式。

（2）需求变更分析：静态演化往往是由于用户需求变化、系统运行出错和运行环境发生改变等原因所引起的。需要找出新的软件需求与原有的差异。

（3）演化计划：分析原系统，确定演化范围和成本，选择合适的演化计划。

（4）系统重构：根据演化计划对系统进行重构，使之适应当前的需求。

（5）系统测试：对演化后的系统进行测试，查找其中的错误和不足之处。

9.2软件架构演化方式的分类

- 1.软件架构静态演化
  - 静态演化的原子演化操作
    - 一次完整软件架构演化过程可以看作是经过一系列原子演化操作组合而成。
    - 所谓原子演化操作，是指基于UML模型表示的软件架构上，在逻辑语义上粒度最小的架构修改操作。

9.2软件架构演化方式的分类

- 1.软件架构静态演化

- 静态演化的原子演化操作

(1) 与可维护性相关的架构演化操作

AMD（Add Module Dependence）	增加模块间的依赖关系
RMD（Remove Module Dependence）	删除模块间的依赖关系
AMI（Add Module Interface）	增加模块间的接口
RMI（Remove Module Interface）	删除模块间的接口
AM（Add Module）	增加一个模块
RM（Remove Module）	删除一个模块
SM（Split Module）	拆分模块
AGM（Aggregate Modules）	聚合模块

9.2软件架构演化方式的分类

- 1.软件架构静态演化
  - 静态演化的原子演化操作

（2）与可靠性相关的架构演化操作

AMS（Add Message）	在顺序图中增加模块交互消息
RMS（Remove Message）	在顺序图中删除模块交互消息
AO（Add Object）	在顺序图中增加交互对象
RO（Remove Object）	在顺序图中删除交互对象
AF（Add Fragment）	在顺序图中增加消息片段
RF（Remove Fragment）	在顺序图中删除消息片段
CF（Change Fragment）	在顺序图中修改消息片段
AU（Add Use Case）	在用例图中为参与者增加一个可执行用例
RU（Remove Use Case）	在用例图中为参与者删除某个可执行用例
AA（Add Actor）	在用例图中增加参与者
RA（Remove Actor）	在用例图中删除参与者

9.2软件架构演化方式的分类

- 2.软件架构动态演化
- 动态演化的需求
  - （1）软件内部执行所导致的体系结构改变。
    - 例如，许多服务器端软件会在客户请求到达时创建新的组件来响应用户需求；
  - （2）是软件系统外部的请求对软件进行的重配置。
    - 例如，操作系统在升级时无须重新启动，在运行过程中就完成对体系结构的修改。

9.2软件架构演化方式的分类

- 2.软件架构动态演化
  - 动态演化的类型

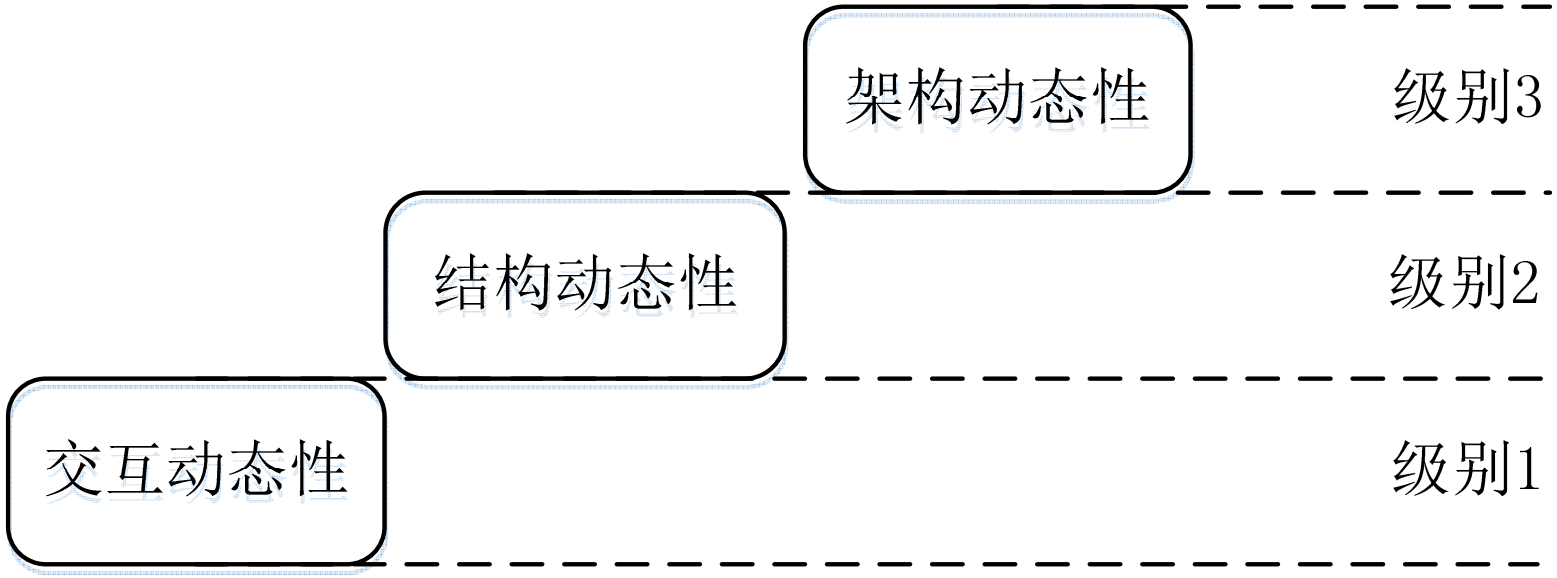


图9.6 软件的三级动态性

9.2软件架构演化方式的分类

- 2.软件架构动态演化
- 动态演化的内容
  - **属性改名**：在运行过程中，用户可能会对非功能指标进行重新定义，如服务响应时间等。
  - **行为变化**：在运行过程中，用户需求变化或系统自身服务质量的调节，都将引发软件行为的变化。如：为了提高安全级别而更换加密算法；将http协议改为https协议。
  - **拓扑结构改变**：如增、删组件，增、删连接件，改变组件与连接件之间的关联关系等。
  - **风格变化**：一般软件演化后其架构风格应当保持不变，如果非要改变软件的架构风格，也只能将架构风格变为其“衍生”风格，如将两层C/S结构调整为三层C/S结构或C/S和B/S的混合结构

9.2软件架构演化方式的分类

- 2.软件架构动态演化
- 动态演化的内容
  - 实现软件架构动态演化的技术主要有两种：
    - 采用动态软件架构（DSA，Dynamic Software Architecture）
    - 进行动态重配置（DR, Dynamic Reconfiguration）

9.2软件架构演化方式的分类

- 2.软件架构动态演化
- 动态软件架构
  - DSA指那些在软件运行时刻会发生变化的体系结构。与静态软件架构相比，DSA的特殊之处在于它的动态性。
  - DSA实施动态演化大体遵循以下四步：
  - （1）捕捉并分析需求变化；
  - （2）获取或生成体系结构演化策略；
  - （3）根据步骤2得到的演化策略，选择适当的演化策略实施演化；
  - （4）演化后的评估与检测。完成这四个步骤还需要DSA描述语言和演化工具的支持。

9.2软件架构演化方式的分类

- 2. 软件架构动态演化
- 动态重配置

一般来说，动态重配置可能涉及到的修改有：

（1）简单任务的相关实现修改；

（2）工作流实例任务的添加和删除；

（3）组合任务流程中的个体修改；

（4）任务输入来源的添加和删除；

（5）任务输入来源的优先级修改；

（6）组合任务输出目标的添加和删除；

（7）组合任务输出目标的优先级修改，等等。

9.3 软件架构演化原则

（I）成本控制原则

原则名称：成本控制原则
原则解释：演化成本要控制在预期的范围之内，也就是演化成本要明显小于重新开发成本。
原则用途：用于判断架构演化的成本是否在可控范围内，以及用户是否可接受。
度量方案：CoE<<CoRD
方案说明：CoE演化成本，CoRD重新开发成本，CoE远小于CoRD最好。

9.3 软件架构演化原则

（2）进度可控原则

原则名称：进度可控原则
原则解释：架构演化要在预期时间内完成，也就是时间成本可控。
原则用途：根据该原则可以规划每个演化过程的任务量；体现一种迭代、递增（持续演化）的演化思想。
度量方案：ttask=|Ttask-T’task|。
方案说明：某个演化任务的实际完成时间(Ttask)和预期完成时间(T’task)的时间差。时间差ttask越小越好。

9.3 软件架构演化原则

（3）风险可控原则

原则名称：风险可控原则
原则解释：架构演化过程中的经济风险、时间风险、人力风险、技术风险和环境风险等必须在可控范围内。
原则用途：用于判断架构演化过程中各种风险是否易于控制?
度量方案：分别检验。
方案说明：不存在 时间风险、经济风险、人力风险和技术风险。

9.3 软件架构演化原则

（4）主体维持原则

原则名称：主体维持原则
原则解释：所有其他因素必须与软件演化协调，开发人员、销售人员、用户必须熟悉软件演化的内容，从而达到令人满意的演化。因此，软件演化的平均增量的增长须保持平稳，保证软件系统主体行为稳定。
原则用途：用于判断架构演化是否导致系统主体行为不稳定。
度量方案：AIG=change(main body)/SIZE(main body)。
方案说明：根据度量动态变更信息[类型、总量、范围]来计算。

9.3 软件架构演化原则

（5）系统总体结构优化原则

原则名称：系统总体结构优化原则
原则解释：架构演化要遵循总体结构优化原则，使得演化之后的软件系统整体结构（布局）更加合理。
原则用途：用于判断系统整体结构是否合理、是否最优?
度量方案：检查系统的整体可靠性和性能指标。
方案说明：判断整体结构优劣的主要指标是系统的可靠性和性能。

9.3 软件架构演化原则

（6）平滑演化原则

原则名称：平滑演化原则（invariant work rate, IWR)

原则解释：在软件系统的生命周期里，软件的演化速率趋于稳定，例如相邻版本的更新率相对固定。

原则用途：用于判断是否存在剧烈架构演化。

度量方案：IWR=变更总量/项目规模。

方案说明：根据度量动态变更信息[类型、总量、范围等]来计算。

9.3 软件架构演化原则

（7）目标一致原则

原则名称：目标一致原则
原则解释：架构演化的阶段目标和最终目标要一致。
原则用途：用于判断每个演化过程是否达到阶段目标，所有演化过程结束是否能达到最终目标。
度量方案：检查阶段目标和最终目标是否一致。
方案说明：可以计算阶段目标的实际达成情况和预期目标的差，这个差越小越好。

9.3 软件架构演化原则

（8）模块独立演化原则[修改局部化原则]

原则名称：模块独立演化原则[修改局部化原则]
原则解释：软件中各模块（相同制品的模块，例如JAVA的某个类或包）自身的演化最好是相互独立，或者至少保证对其他模块的影响比较小、或影响范围比较小。
原则用途：用于判断每个模块自身的演化是否相互独立的。
度量方案：检查模块的修改是否是局部的。
方案说明：可以通过计算修改的影响范围来进行度量。

9.3 软件架构演化原则

（9）影响可控原则

原则名称：影响可控原则
原则解释：软件中一个模块如果发生变更给其他模块带来的影响要在可控范围内，也就是影响范围可预测。
原则用途：用于判断是否存在对某个模块的修改导致大量其他修改的情况。
度量方案：检查影响的范围是否可控。
方案说明：可以通过计算修改的影响范围来进行度量。

9.3 软件架构演化原则

（10）复杂性可控原则

原则名称：复杂性可控原则
原则解释：架构演化必须要控制架构的复杂性，从而进一步保障软件的复杂性在可控范围内。
原则用途：用于判断演化之后的架构是否易维护、易扩展、易分析、易测试等。
度量方案：CC<某个阈值。
方案说明：CC增长可控。

9.3 软件架构演化原则

（11）有利于重构原则

原则名称：有利于重构原则
原则解释：架构演化要遵循有利于重构原则，使得演化后的软件架构更便于重构。
原则用途：用于判断架构易重构性是否得到提高?
度量方案：检查系统的复杂度指标。
方案说明：系统越复杂越不容易重构。

9.3 软件架构演化原则

（12）有利于重用原则

原则名称：有利于重用原则
原则解释：架构演化最好能维持、甚至提高整体架构的可重用性。
原则用途：用于判断整体架构可重用性是否遭到破坏?
度量方案：检查模块自身的内聚度、模块之间的耦合度。
方案说明：模块的内聚度越高、该模块与其他模块之间的耦合度越低越容易重用。

9.3 软件架构演化原则

（13）设计原则遵从性原则

原则名称：设计原则遵从性原则
原则解释：架构演化最好不能与架构设计原则冲突。
原则用途：用于判断架构设计原则是否遭到破坏.
【架构设计原则是好的设计经验总结，要保障得到充分使用】
度量方案：RCP=|CDP|/|DP|
方案说明：冲突的设计原则集合(CDP)和总的设计原则集合(DP)的比较，RCP越小越好。

9.3 软件架构演化原则

（14）适应新技术原则

原则名称：适应新技术原则
原则解释：软件要独立于特定的技术手段，这样就能够让软件运行于不同平台。
原则用途：用于判断架构演化是否存在对某种技术依赖过强的情况。
度量方案：TI=1-DDT，其中DDT=|依赖的技术集合|/|用到的技术合集|。
方案说明：根据演化系统对关键技术的依赖程度进行度量。

9.3 软件架构演化原则

（15）环境适应性原则

原则名称：环境适应性原则
原则解释：架构演化后的软件版本能够比较容易适应新的硬件环境与软件环境。
原则用途：用于判断架构在不同环境下是否仍然可使用，或者环境配置容易。
度量方案：硬件/软件兼容性。
方案说明：结合软件质量中兼容性指标进行度量。

9.3 软件架构演化原则

（16）标准依从性原则

原则名称：标准依从性原则
原则解释：架构演化不会违背相关质量标准【国际标准、国家标准、行业标准、企业标准等】。
原则用途：用于判断架构演化是否具有规约性，是否有章可循；而不是胡乱的或随意的演化。
度量方案：需要人工判定
方案说明：判断架构演化不会违背相关质量标准【国际标准、国家标准、行业标准、企业标准等】。

9.3 软件架构演化原则

（17）质量向好原则

原则名称：质量向好原则
原则解释：通过演化使得所关注的某个质量指标或某些质量指标的综合效果变得更好或者更满意。例如可靠性提高了。
原则用途：用于判断架构演化是否导致某些质量指标变得很差。
度量方案：EQI>SQ
方案说明：演化之后的质量（EQI）比原来的原来的质量（SQ）要好

9.3 软件架构演化原则

（18）适应新需求原则

原则名称：适应新需求原则[new requirement adaptability]

原则解释：架构演化要很容易适应新的需要变更：架构演化不能降低原有架构适应新需求的能力；架构演化最好可以提高适应新需求的能力。

原则用途：用于判断演化之后的架构是否降低了架构适应新需求的能力。

度量方案：RNR=|ANR|/|NR|。

方案说明：适应的新需求集合(ANR)和实际新需求集合(NR)的比较，RNR越小越好。

9.4软件架构维护

- 软件架构的开发和维护是基于架构软件生命周期中的关键环节。
- 维护需要对软件架构的演化过程进行追踪和控制，保障软件架构的演化过程能够满足需求。
- 软件架构维护过程一般涉及到架构知识管理、架构修改管理和架构版本管理等内容

9.4软件架构维护

- 1.软件架构知识管理

- 架构知识=架构设计+架构设计决策。
- 架构知识管理是对架构设计中所隐含的决策来源进行文档表示，进而在架构维护过程中帮助维护人员对架构的修改进行完善的考虑，并能够为其他软件架构的相关活动提供参考。

9.4软件架构维护

- 1.软件架构知识管理

- 架构知识管理的含义

架构知识管理侧重于软件开发和实现过程所涉及到的架构静态演化，从架构文档等信息来源中捕捉架构知识，进而提供架构的质量属性及其设计的根据进行记录和评价。

- 架构知识管理的现状

当前尚无实用的架构知识整理策略，构建架构的涉众（即拥有架构知识的人）通常不会使用文档记录架构知识。

9.4软件架构维护

- **2.**软件架构修改管理

软件架构修改管理中，一个主要的做法就是建立一个隔离区域，保障该区域中任何修改对其他部分的影响比较小，甚至没有影响。为此，需要明确修改规则、修改类型和可能的影响范围和副作用等。

9.4软件架构维护

- **3.** 软件架构版本管理

软件架构演化的版本管理为软件架构演化的版本管理、控制、利用和评价等提供了可靠的依据，并为架构演化量化测量奠定了基础。

9.5本章小结

（1）软件架构在构建之后不可避免的要进行修改，进入演化和维护阶段；

（2） 架构的演化和维护既存在于软件开发期间，也存在于软件维护期间；

（3）在软件运行期间所进行的架构演化为架构动态演化，较之架构静态演化更加的困难而重要；

（4）软件的动态性包括三个层次：数据层次的动态性、结构层次动态性和架构层次动态性，其中数据层次的动态性是软件常见的功能；

（5）结构层次的动态性即为软件系统在执行过程中对组件的动态重配置，通过一定的机制将需要演化的组件从系统中脱离出来，完成修改，实现系统的无缝升级。